New Programming Paradigms

- Don Batory
- Ira Baxter
- Karl Crary
- Premkumar Devanbu
- Tzilla Elrad
- Paul Hudak
- Ralph Johnson
- Gregor Kiczales
- Sriram Krishnamurthi
- James Larus
- Karl Lieberherr
- Tommy McGuire
- Michael Mislove
- Benjamin Pierce
- Joy Reed
- Spencer Rugaber
- Charles Simonyi
- Frank Sledge
- Doug Smith
- Kurt Stirewalt
- Janos Sztipanovitzs

software is inherently complex

- and we want to build more and more complex systems
- we deal with complexity by
  - abstraction and (de)composition
  - aka ‘separation of concerns’
- progress in abstraction and (de)composition is basic to our field
  - we’ve made progress handling some concerns before
  - but more progress is needed
  - there is an opportunity to make significant progress

(a sea of) concerns

- domain knowledge, environment knowledge…
- mobility, adaptability, testability, resilience, security, functionality, distribution, real-time constraints, cost constraints, time constraints, fault-tolerance, verifiability, standards conformance, scale
- architectures, algorithms, data structures

what good abstraction and (de)composition means

- capture the individual concern
  - in a clear and natural form (a means of expression)
  - well localized
  - with a clear abstraction
    - ‘interface’ to the rest of the system
- be able to understand the rest of the system in terms of the abstraction of the concern
- be able to (automatically) compose concerns to form the whole

opportunity

- we now have more kinds of (de)composition mechanisms:
  - hierarchical (objects, procedures…)
  - crosscutting (aspects, subjects…)
- this can enable using different kinds of abstraction and decomposition frameworks together together in powerful new ways

vision:
multi-faceted software development

rigorous, top-down, prototype, incremental, methodological…

functionality, real-time, domain…

a scientifically based practice of developing and expressing each concern

in its most appropriate form,

operating on each concern,

and combining to produce whole

DSL, GPL, type system, UML, linear logic...

implementation, design…

analysis, checking…

compiling, RPC, refinement, weaving…

scope of vision

- something old, something new…
- we have pieces of this today
  - following four slides are examples of what we have today that fits this vision, and suggests further research
- recent results should enable dramatic progress on this vision in next 10 years

e.g. (1) – model-based computing

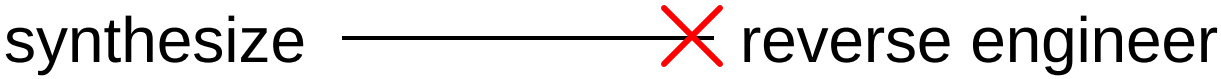
- for example
  - a model captures timing constraints among components
  - checks that such constraints can be specified
  - generates code

e.g. (2) – UML

- class diagram
  - captures structure of system
- interaction diagrams
  - capture different sequences of operations
- generate code, edit code, generate model…

e.g. (3) – Bold Stroke in AOP experiment

- ordinary OOP to capture component functionality
  - how component produces output data from input
- AOP to capture event, data flow and execution aspects
  - event flow rules, data flow rules, update rules
- compiler weaves aspects with components
- formal reasoning about global state and end-to-end properties



e.g. (4) – partial spec plus checking

- ordinary code to implement system
- type system
  - ensure data abstractions respected
- temporal behavior specifications
  - e.g. file opened before reading, grab lock before accessing structure